11 コレクション

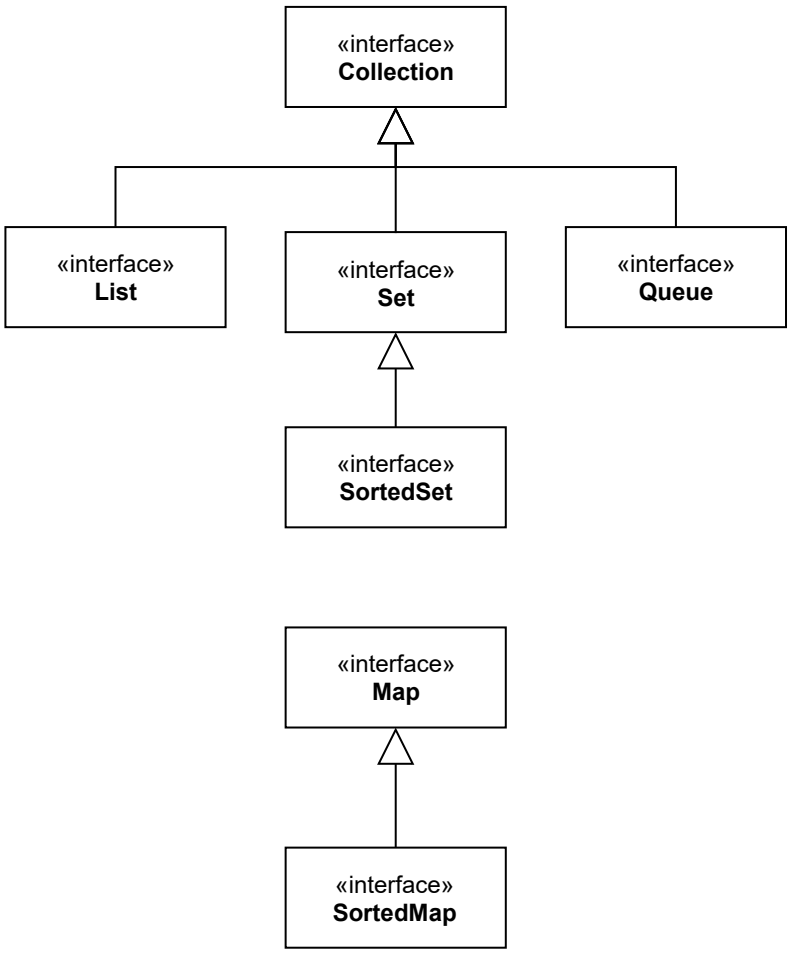
作成者：GT F
最終更新日：2019-08-20

基本概念

コレクションフレームワークとは複数のオブジェクトをまとめ管理する仕組みです。これらのクラスおよびインターフェースは`java.util`パッケージで提供されます。共通コレクションに対して以下共通操作があります。

1. 要素を追加する
2. 要素を削除する
3. 要素を検索する
4. 要素を取り出す

コレクションに格納するオブジェクトを**要素**あるいは**エレメント**と呼びます。java.utilパッケージに、以下コレクションクラスを提供します。



ジェネリック

ジェネリンク（Generics）とは、クラスやインスタンス、メソッド等の「型」をパラメータ化する機能です。

ダイヤモンド演算子

ダイヤモンド`<>`。

ジェネリンクを使用して変数を宣言する構文：型<データ型> 変数名;

```
List<String> array = new ArrayList<>();
```

コレクション操作

List

リストとはサイズ変更可能な配列であり。インスタンスListを実現する標準クラスは３つあります。

クラス名	特徴
ArrayList	検索は高速、追加又削除は低速。並列アクセス不可。
LinkedList	検索は低速、追加又削除高速。並列アクセス不可。
Vector	ArrayListと似ていますが、並列アクセス可能

リストはジェネリック対応していますので、使用方法はダイアモンド演算子`<>`を使用して、インスタンス化する。

```
List<String> list1 = new ArrayList<>();
List<String> list2 = new LinkedList<>();
List<String> list3 = new Vector<>();
```

1. 要素を追加する

例.`add(T data)`メソッドを利用して、要素を追加

```
public static void main(String...args) {
    List<String> datas = new ArrayList<>();
    datas.add("user1");
}
```

2.要素を削除する

指定されたオブジェクトを削除し、またすべてエレメントを削除するメソッドを提供します。

```
public static void main(String...args) {
    List<String> datas = new ArrayList<>();
```

```
3       datas.add("user1");
4       datas.remove("user1"); // user1 を削除する
5       datas.clear(); // すべてのエレメントを削除する
6   }
```

⚠ リストのエレメントを削除するメソッドはオブジェクトのequalsメソッドを呼び出して比較します。一致する場合、削除します。

3.要素を検索する

指定されたオブジェクトは既にリストに存在有無をチェックします。なお、存在する場合、オブジェクトのインデックスを取得可能です。

```
1   public static void main(String... args) {
2       List<String> datas = new ArrayList<>();
3       datas.add("user1");
4       System.out.println(datas.contains("user1"));// true：存在する
5       System.out.println(datas.indexOf("user2"));// -1
6   }
```

4.要素を取り出す

指定されたインデックスのオブジェクトを取得します。オブジェクトが存在しない場合、`null`を返します。

```
1   public static void main(String... args) {
2       List<String> datas = new ArrayList<>();
3       datas.add("user1");
4       System.out.println(datas.get(0));// user1
5   }
```

5.List & 配列

List TO 配列

```
1   public static void main(String...args) {
2       List<String> list = new ArrayList<>();
3       list.add("item 1");
4       String[] datas = list.toArray(new String[0]); // [0]サイズは０でも問題がありません。
5   }
```

配列 TO List

```
1   public static void main(String...args) {
2       String[] datas = new String[]{"item1", "item2", "item3"};
3       List<String> list = Arrays.asList(datas);
4   }
```

6.ソート

標準クラス`java.util.Collections`ソートメソッドを提供します。ディフォルトは昇順でソートします。

```
public static void main(String... args) {
    List<String> datas = new ArrayList<>();
    datas.add("u1");
    datas.add("u3");
    datas.add("u2");
    System.out.println(datas); // [u1, u3, u2]
    Collections.sort(datas);
    System.out.println(datas); // [u1, u2, u3]
}
```

ソート順をカスタマイズする可能です。この場合、`Comparator`クラスを継承して、比較メソッドを実装した後、`java.util.Collections::sort`メソッドの第２引数として渡します。

```
public static void main(String... args) {
    List<String> datas = new ArrayList<>();
    datas.add("u1");
    datas.add("u3");
    datas.add("u2");
    System.out.println(datas); // [u1, u3, u2]

// 匿名クラス
//  Collections.sort(datas, new Comparator<String>() {
//      @Override
//      public int compare(String t, String t1) {
//          return t2.compareTo(t1);
//      }
//  });
    // このラムダ式をよく理解してください。
    Collections.sort(datas, (t1, t2) -> {
        return t2.compareTo(t1);
    });
    System.out.println(datas); // [u3, u2, u1]
}
```

Map

データをキーと値のペアを管理するデータ構造はマップです。マップのキーは一意であり、値は重複可能です。代表のマップクラスは以下３つあります。

クラス名	特徴
HashMap	よく利用するマップです

LinkedHashMap	エレメントの順が維持されています
TreeMap	挿入したエレメントは自動的に昇順でソートします。

マップを初期するとき、鍵のタイプと値のタイプ指定する必要です。

```
Map<String, Object> map = new HashMap<>();
```

1. 要素を追加する

マップに要素を追加する為、キーと値を用意して挿入します。マップのキーは一意の為、２回同じキーを挿入する場合、前の値を上書されます。

```
public static void main(String... args) {
    Map<String, String> map = new HashMap<>();
    map.put("key1", "value1");
    System.out.println(map);//{key1=value1}
    map.put("key1", "value2");
    System.out.println(map);//{key1=value2} 同じキーの場合、値を上書き
}
```

2.要素を削除する

マップに既に存在する要素を削除可能です。指定されたキーに対する要素を削除は以下メソッドを呼び出します。

```
public static void main(String... args) {
    Map<String, String> map = new HashMap<>();
    map.put("key1", "value1");
    System.out.println(map);//{key1=value1}
    map.remove("key1");// キーと値のペア両方を削除します。
    System.out.println(map);//{}
}
```

3.要素を検索する

指定されたキーと値のペアが存在有無チェックする可能です。

```
public static void main(String... args) {
    Map<String, String> map = new HashMap<>();
    map.put("key1", "value1");
    System.out.println(map.containsKey("key1")); // true
    System.out.println(map.containsKey("key2")); // false

}
```

4.要素を取り出す

指定されたキーで値を取得する。キーが存在する場合、対応する値を返します。キーが存在しない場合、`null`を返します。なお、キーが存在しない場合、ディフォルト値返却するように設定可能です。

```
1  public static void main(String... args) {
2      Map<String, String> map = new HashMap<>();
3      map.put("key1", "value1");
4      map.put("key3", null);
5
6      System.out.println(map.get("key1")); // value1
7      System.out.println(map.get("key2")); // null
8      System.out.println(map.get("key3")); // null
9      System.out.println(map.getOrDefault("key3", "Empty")); // null
10     System.out.println(map.getOrDefault("key4", "Empty")); // Empty
11 }
```

空判断

メソッド`isEmpty()`を使用して、リストまたマップは空ではないかを判断する。

```
1  public static void main(String... args) {
2      List<String> list = new ArrayList<>();
3      System.out.println(list.isEmpty()); // true
4      Map<String, String> map = new HashMap<>();
5      System.out.println(map.isEmpty()); // true
6  }
```

※list.size() == 0 を絶対使用しないで、阿呆みたい...

拡張for文 Ⅱ

List

拡張for文(foreach)を用い、リストを簡単にループできます。

```
1  public static void main(String... args) {
2      List<String> datas = new ArrayList<>();
3      datas.add("u1");
4      datas.add("u3");
5      datas.add("u2");
6      // リストに各エレメントをアクセスする
7      for (String data : datas) {
8          System.out.println(data);
9      }
10 }
```

Map

マップクラスでも拡張for文使用可能ですが、ループのローカル変数はキーと値のペアです。

```
1  public static void main(String... args) {
2      Map<String, String> map = new HashMap<>();
3      map.put("key1", "value1");
4      map.put("key3", null);
5      // マップのキーと値をペアでループする
6      for (Map.Entry<String, String> entry : map.entrySet()) {
7          System.out.println("key=" + entry.getKey() + "; value=" + entry.getValue());
8      }
9  }
```

iterator

反復子（イテレータ）を用い、コレクションの各エレメントアクセス可能。イテレータは以下メソッドを保有します。

1. hasNext 次のエレメントが有無を判断する。
2. next 次のエレメントへ移動する。

List

リストから`iterator()`メソッドを呼び出す、イテレータを取得します。次のエレメントが存在する場合、次へ移動します。

```
1  public static void main(String... args) {
2      List<String> datas = new ArrayList<>();
3      datas.add("u1");
4      datas.add("u3");
5      datas.add("u2");
6      Iterator<String> it = datas.iterator();
7      while (it.hasNext()) {
8          System.out.println(it.next());
9      }
10 }
```

Map

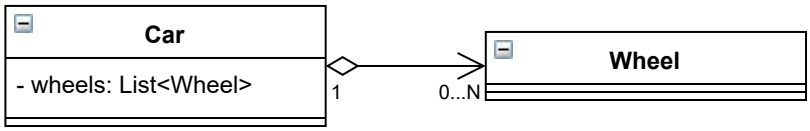
マップの.`entryset()`からさらに`iterator()`メソッドを呼び出し、イテレータを取得します。次のエレメントが存在する場合、次へ移動します。

```
1  public static void main(String... args) {
2      Map<String, String> map = new HashMap<>();
3      map.put("key1", "value1");
```

```
4       map.put("key3", null);
5       Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();
6       while (it.hasNext()) { //
7           Map.Entry<String, String> entry = it.next();
8           System.out.println("key=" + entry.getKey() + "; value=" + entry.getValue());
9       }
10  }
```

集約関係 Ⅱ

OOP - その1 からクラス関係を１：１を定義したですが、今回コレクションクラスを用い、１：N関係を定義します。




```
1   // Wheel.java
2   public class Wheel {
3   }
4   // Car.java
5   public class Car {
6       private List<Wheel> wheels;
7   }
```

代表的なコレクション

代表的なコレクションの実装における特性を下記表にまとめます。

クラス	インターフェース	一意	順序付け
ArrayList	List	N	Y
LinkedList	List	N	Y
Vector	List	N	Y
HashSet	Set	Y	N
LinkedHashSet	Set	Y	N
TreeSet	Set	Y	N

PriorityQueue	Queue	N	N
Stack	Stack	N	Y
HashMap	Map	N	N
LinkedHashMap	Map	N	N
Hashtable	Map	N	N
TreeMap	Map	N	N

リストのリスト

多次元の配列を作成可能ですが、多次元のリストも作成可能。

```
List<List<String>> list2d = new ArrayList<>();
List<String> row1 = new ArrayList<>();
row1.add("1");
row1.add("2");
list2d.add(row1);
System.out.println(list2d);//[[1, 2]]
```

マップのリスト

マップとリストも組み合わせ可能です。

```
public static void main(String... args) {
    Map<String, List<String>> map = new HashMap<>();
    List<String> list = new ArrayList<>();
    list.add("2019/06/01");
    list.add("2019/06/02");
    list.add("2019/06/03");
    map.put("201906", list);
    //{201906=[2019/06/01, 2019/06/02, 2019/06/03]}
    System.out.println(map);
}
```

質問

質問１：駅クラス（Station）、路線クラス（Line）を作成して。路線をインスタンス化してオブジェクト「山手線」を作成してください。「山手線」に各駅情報を追加してください。

質問２：質問１に作成した山手線の駅を`foreach`文を繰り返して、新橋駅は路線内であるかを判断する。

質問３：10両編成している通勤電車、各車の定員は5人です。31人の乗車の状況をプログラミングしてください。（for文で旅客作成しても構わない）

いいね　1 番に「いいね」しましょう

ラベルがありません